# Detailed Instructions for Your Friend - Model Setup (No Code, Just Steps)

## 🎯 Your Mission

Download pre-trained AI models from Qualcomm AI Hub and prepare them for the CampusGuard app. You'll be working on the **Snapdragon X Elite laptop** provided at the hackathon.

**Time Budget: 3-4 hours**

---

## Part 1: Initial Setup (30 minutes)

### Step 1: Set Up Your Workspace

1. **Open the Snapdragon X Elite laptop** they provide at the hackathon
2. **Create a project folder:**
   - Open File Explorer (Windows) or Finder (Mac)
   - Navigate to Documents
   - Create new folder: `CampusGuard_Models`
   - Inside that, create subfolders:
     - `downloaded_models`
     - `test_scripts`
     - `documentation`

### Step 2: Install Python (if not already installed)

1. Open Command Prompt (Windows) or Terminal (Mac/Linux)
2. Type: `python --version` and press Enter
3. **If you see version 3.8 or higher:** You're good, skip to Step 3
4. **If you see an error or older version:**
   - Go to: https://www.python.org/downloads/
   - Download Python 3.10 or 3.11
   - Run installer
   - ✅ **IMPORTANT:** Check "Add Python to PATH" during installation
   - Restart your terminal after installation

### Step 3: Install Required Python Libraries

1. In your terminal, type these commands one by one:

None

```
pip install onnxruntime
pip install opencv-python
pip install numpy
pip install pillow
```

2. Wait for each to finish installing (2-3 minutes total)
3. If you see "Successfully installed..." - you're ready!

---

## Part 2: Qualcomm AI Hub Account & Navigation (15 minutes)

### Step 1: Create Your Account

1. **Open browser** and go to: **https://aihub.qualcomm.com/**
2. Click **"Sign In"** (top right corner)
3. Click **"Create Account"**
4. Fill in:
    - Email (use your real email)
    - Password
    - Name
    - Accept terms
5. Click **"Sign Up"**
6. **Check your email** for verification link
7. Click the link to verify
8. **Log back in** to AI Hub

### Step 2: Understand the AI Hub Interface

Once logged in, you'll see:

- **Top navigation bar:** Models | Compute | Docs | Community
- **Search bar:** To find specific models
- **Model cards:** Each shows a different AI model
- **Filters:** On left side (Platform, Task, etc.)

## Part 3: Download Model #1 - YOLOv8 for Person Detection (45 minutes)

### Step 1: Find the YOLOv8 Model

1. In the **search bar at top**, type: `YOLOv8`
2. Press Enter
3. You'll see several YOLO models appear
4. Look for one that says:
   - **"YOLOv8-Det"** or **"YOLOv8 Detection"** or **"YOLO-NAS"**
   - Should mention "object detection" or "person detection"
5. **Click on the model card** to open details

### Step 2: Review Model Information

On the model page, look for these key details and **write them down**:

- **Model Name:** (e.g., "YOLOv8n-Detection")
- **Input Size:** (usually 640x640 or similar)
- **Input Format:** (look for NCHW or NHWC)
- **Supported Platforms:** Make sure it says "Snapdragon 8 Elite"
- **Task:** Should say "Object Detection"

### Step 3: Download the Model

1. Scroll down to **"Download"** section
2. You'll see platform options:
   - Snapdragon 8 Elite ✅ **SELECT THIS ONE** (for your phone)
   - Snapdragon X Elite
   - Cloud AI 100
3. Click **"Download"** for Snapdragon 8 Elite
4. You might see format options:
   - **ONNX** ✅ **SELECT THIS**
   - TFLite
   - PyTorch
5. Click download and wait (file is usually 5-50 MB)
6. **Save to:** `CampusGuard_Models/downloaded_models/`

### Step 4: Extract and Organize

1. Once downloaded, you'll have a `.zip` file

2. **Right-click → Extract All**
3. Extract to: `CampusGuard_Models/downloaded_models/yolov8/`
4. Inside you should see:
    - `model.onnx` (or similar name)
    - `model_card.md` or README
    - Maybe sample code files
5. **Rename** the `.onnx` file to: `yolov8_person_detection.onnx`

### Step 5: Document Model Specifications

1. Open the `model_card.md` or README file
2. Create a new text file: `model_specs.txt`
3. **Write down these critical details:**

None

```
MODEL 1: YOLOv8 Person Detection
================================
File name: yolov8_person_detection.onnx
File size: [check the actual size]

INPUT SPECIFICATIONS:
- Input name: [look in model card - usually "images" or "input"]
- Input shape: [e.g., 1, 3, 640, 640]
- Format: [NCHW or NHWC]
- Data type: [float32 or int8]
- Value range: [0-1] or [0-255]
- Preprocessing needed: [resize to 640x640, normalize, etc.]

OUTPUT SPECIFICATIONS:
- Number of outputs: [usually 1 or 3]
- Output names: [e.g., "output0" or "boxes, scores, classes"]
- Output shapes: [write what you see]
- What each output means:
  - Output 0: Bounding boxes [x, y, width, height]
  - Output 1: Confidence scores
  - Output 2: Class IDs (0 = person in COCO dataset)

NOTES:
```

```
- Person class ID is: 0 (from COCO dataset)
- Confidence threshold recommended: 0.5 or higher
```

---

## Part 4: Download Model #2 - MoveNet for Pose Detection (45 minutes)

### Step 1: Find MoveNet

1. Back on AI Hub homepage
2. **Search:** `MoveNet` or `pose estimation`
3. Look for:
   - **"MoveNet Lightning"** (faster, good for real-time) ✅ **PREFER THIS**
   - or "MoveNet Thunder" (more accurate, slower)
4. Click on the model card

### Step 2: Download MoveNet

1. Review the model page (same as before)
2. Select platform: **Snapdragon 8 Elite**
3. Select format: **ONNX**
4. Click Download
5. Save to: `CampusGuard_Models/downloaded_models/`

### Step 3: Extract and Organize

1. Extract the `.zip` file
2. Extract to: `CampusGuard_Models/downloaded_models/movenet/`
3. **Rename** the `.onnx` file to: `movenet_pose_detection.onnx`

### Step 4: Document MoveNet Specifications

Add to your `model_specs.txt`:

None

```
MODEL 2: MoveNet Pose Estimation
=================================
File name: movenet_pose_detection.onnx
File size: [check actual size]

INPUT SPECIFICATIONS:
- Input name: [usually "input" or "image"]
- Input shape: [typically 1, 192, 192, 3 for Lightning]
- Format: [NHWC - different from YOLO!]
- Data type: [int32 or float32]
- Value range: [0-255] typically
- Preprocessing: [resize to 192x192]

OUTPUT SPECIFICATIONS:
- Output shape: [usually 1, 1, 17, 3]
- What it means:
  - 17 = number of body keypoints
  - 3 = [y_coordinate, x_coordinate, confidence]

KEYPOINT MAPPING (17 points):
0: nose
1-2: left eye, right eye
3-4: left ear, right ear
5-6: left shoulder, right shoulder
7-8: left elbow, right elbow
9-10: left wrist, right wrist
11-12: left hip, right hip
13-14: left knee, right knee
15-16: left ankle, right ankle

NOTES:
- Coordinates are normalized [0, 1]
- Multiply by image size to get pixel positions
- Confidence > 0.3 is generally reliable
```

## Part 5: Test Models Work (1 hour)

Now you need to verify these models actually run.

### Step 1: Create Test Script

1. Open **Cursor** (or any code editor)
2. Create new file: `test_models.py`
3. Save it in: `CampusGuard_Models/test_scripts/`

### Step 2: Use Cursor AI to Generate Test Code

In Cursor, use the AI assistant and give this prompt:

None

```
I need a Python script to test two ONNX models:

1. yolov8_person_detection.onnx
   - Input: [1, 3, 640, 640] float32
   - Task: Person detection

2. movenet_pose_detection.onnx
   - Input: [1, 192, 192, 3] float32
   - Task: Pose estimation

The script should:
- Load both models using onnxruntime
- Print input/output shapes
- Run inference on dummy data
- Verify both models work without errors
- Print "SUCCESS" if everything passes

Model files are in: ../downloaded_models/yolov8/ and
../downloaded_models/movenet/
```

### Step 3: Run the Test

1. Open terminal in the `test_scripts` folder

2. Run: `python test_models.py`

3. **Expected output should show:**
    - "Loading YOLOv8..."
    - "✅ YOLOv8 loaded successfully"
    - Model input/output details
    - "Loading MoveNet..."
    - "✅ MoveNet loaded successfully"
    - "SUCCESS - All models working!"
4. **If you see errors:**
    - Read the error message
    - Most common: file path wrong
    - Check that `.onnx` files are in the right folders
    - Check file names match exactly

---

# Part 6: Create Anomaly Detection Logic (1.5 hours)

This is the "intelligence" that decides if something is suspicious.

## Step 1: Design the Detection Rules

Before coding, write down the logic in plain English:

**Create a file:** `anomaly_rules.txt`

None

```
CAMPUSGUARD ANOMALY DETECTION RULES
====================================

RULE 1: PERSON DOWN (Fallen Person)
- Trigger: Person bounding box is in bottom 25% of frame
- AND: Person's height is less than width (lying down)
- Confidence score: 0.85
- Event type: "Person Down - Possible Emergency"

RULE 2: UNUSUAL POSE (from MoveNet)
- Trigger: Person's hip keypoints are lower than ankle keypoints
  (indicates person is on ground)
```

```
- OR: Both shoulders have low confidence but hips are visible
  (person bent over or crouching)
- Confidence score: 0.75
- Event type: "Unusual Body Position Detected"

RULE 3: RAPID MOVEMENT
- Trigger: Person bounding box moves > 100 pixels between frames
- AND: Movement is sustained over 3+ frames
- Confidence score: 0.70
- Event type: "Rapid Movement - Running Detected"

RULE 4: CROWD FORMATION
- Trigger: 5+ people detected in frame suddenly
- AND: They weren't there in previous frame
- Confidence score: 0.65
- Event type: "Sudden Crowd Gathering"

RULE 5: LOITERING
- Trigger: Same person bounding box stays in similar position for
30+ seconds
- AND: Very minimal movement (< 20 pixels)
- Confidence score: 0.60
- Event type: "Person Loitering in Area"

FINAL DECISION:
- Take the MAXIMUM confidence score from all triggered rules
- If max confidence > 0.70 → Alert user
- If 0.50 - 0.70 → Log but don't alert (borderline)
- If < 0.50 → Ignore
```

### Step 2: Ask Cursor to Implement Rules

Create new file: `anomaly_detector.py`

Give Cursor this prompt:

None

```
Create a Python class called CampusGuardAnomalyDetector that:

1. Loads two ONNX models:
   - YOLOv8 for person detection (input: 1,3,640,640)
   - MoveNet for pose estimation (input: 1,192,192,3)

2. Has a method: detect_anomaly(frame) that:
   - Takes a camera frame (numpy array)
   - Runs both models on it
   - Applies these 5 anomaly rules: [paste your anomaly_rules.txt here]
   - Returns: (is_anomalous: bool, confidence: float, event_type: str)

3. Tracks previous frames to detect:
   - Rapid movement (compare current boxes to previous)
   - Loitering (person staying in same spot)

4. Helper methods:
   - preprocess_for_yolo(frame) → converts to [1,3,640,640]
   - preprocess_for_movenet(frame) → converts to [1,192,192,3]
   - parse_yolo_output() → extracts boxes, scores, classes
   - parse_movenet_output() → extracts 17 keypoints
   - check_person_down(box, keypoints) → applies Rule 1 & 2
   - check_rapid_movement(current_boxes, prev_boxes) → applies Rule 3
   - check_crowd(boxes) → applies Rule 4

Make it clean, well-commented, and production-ready.
```

### Step 3: Test the Anomaly Detector

1. Create `test_anomaly_detector.py`

2. Ask Cursor to generate test code that:

    ○ Loads the anomaly detector

    - Creates fake test frames:
        - Normal scene
        - Person lying down scene
        - Running person scene
    - Runs detection on each
    - Prints results
3. Run: `python test_anomaly_detector.py`

4. Verify you see different anomaly types detected correctly

---

## Part 7: Prepare for Android Integration (30 minutes)

### Step 1: Copy Models to Android Assets

1. Create a folder: `CampusGuard_Models/for_android/`
2. **Copy these files:**
    - `yolov8_person_detection.onnx`
    - `movenet_pose_detection.onnx`
3. Create a README in that folder:

None

```
MODELS FOR ANDROID APP
======================

Files to copy to: app/src/main/assets/

1. yolov8_person_detection.onnx (person detection)
2. movenet_pose_detection.onnx (pose estimation)

These files are already optimized for Snapdragon 8 Elite.
No further conversion needed.
```

### Step 2: Document Integration Steps

Create: `android_integration_guide.txt`

None

```
HOW TO USE THESE MODELS IN ANDROID
===================================

STEP 1: Copy .onnx files to app/src/main/assets/

STEP 2: In Android (Kotlin), load models like this:
val modelBytes =
context.assets.open("yolov8_person_detection.onnx").readBytes()
val session = ortEnvironment.createSession(modelBytes)

STEP 3: Preprocess camera frames:
- Resize to 640x640 for YOLO
- Convert to float array
- Normalize to [0, 1]
- Format: NCHW (channels first)

STEP 4: Run inference:
val inputs = mapOf("images" to inputTensor)
val outputs = session.run(inputs)

STEP 5: Parse outputs:
- YOLO returns: boxes, scores, classes
- Filter for class=0 (person)
- Keep only confidence > 0.5

STEP 6: Apply anomaly rules:
- Check if box is in bottom 25% of frame
- Check for rapid movement vs previous frame
- Return anomaly flag to UI

KEY DIFFERENCES FROM PYTHON:
- No OpenCV (use Android Bitmap instead)
- No NumPy (use FloatArray)
- Preprocessing is manual in Kotlin
```

### Step 3: Create Example Preprocessing Code

Ask Cursor to generate this in a new file `kotlin_preprocessing_example.kt`:

None

```
Generate Kotlin code (just as reference, don't need to run) that
shows:

1. How to convert Android Bitmap to Float Array for YOLO
   - Resize to 640x640
   - Extract RGB values
   - Normalize to [0,1]
   - Arrange in NCHW format [1, 3, 640, 640]

2. How to convert Bitmap to Float Array for MoveNet
   - Resize to 192x192
   - Keep in NHWC format [1, 192, 192, 3]
   - Values in [0, 255] range

3. Helper function to draw bounding boxes on Bitmap
4. Helper function to draw pose keypoints on Bitmap

Make it copy-paste ready for Android development.
```

---

## Part 8: Final Deliverables Checklist

Before calling it done, make sure you have:

### ✅ File Structure Check

None

```
CampusGuard_Models/
├── downloaded_models/
│   ├── yolov8/
│   │   └── yolov8_person_detection.onnx
│   └── movenet/
│       └── movenet_pose_detection.onnx
├── test_scripts/
```

```
|   ├── test_models.py
|   ├── anomaly_detector.py
|   └── test_anomaly_detector.py
├── documentation/
|   ├── model_specs.txt
|   ├── anomaly_rules.txt
|   └── android_integration_guide.txt
└── for_android/
    ├── yolov8_person_detection.onnx
    ├── movenet_pose_detection.onnx
    └── README.txt
```

### ✅ Verification Steps

- [ ] Both models downloaded and renamed
- [ ] Model specs documented with input/output details
- [ ] Test script runs without errors
- [ ] Anomaly detector class created and tested
- [ ] Android integration guide written
- [ ] Models copied to `for_android/` folder
- [ ] Ready to share with Android developer (your other friend)

---

# Part 9: Share with Team

## What to Send to Android Developer

Create a **zip file** containing:

- `for_android/` folder (with both .onnx files)
- `android_integration_guide.txt`
- `model_specs.txt`
- `kotlin_preprocessing_example.kt`

Send message:

None

```
Hey! Models are ready.

I've attached:
1. Two ONNX files - copy these to app/src/main/assets/
2. Integration guide with exact preprocessing steps
3. Example Kotlin code for reference

Key info:
- YOLO input: [1, 3, 640, 640] float32, range [0,1]
- MoveNet input: [1, 192, 192, 3] float32, range [0,255]
- Person class ID = 0
- Confidence threshold: 0.5+

The anomaly detection logic is in anomaly_detector.py - you'll
need to translate this to Kotlin.

Let me know if you need clarification on anything!
```

---

# Troubleshooting Guide

## Issue: Can't find models on AI Hub

**Solution:**

- Try searching: "object detection" or "pose estimation"
- Filter by Platform: Snapdragon 8 Gen 3 or 8 Elite
- Look in "Computer Vision" category

## Issue: Download button is grayed out

**Solution:**

- Make sure you're logged in
- Some models require accepting license terms first
- Check if your account is verified

## Issue: Models won't load in Python

**Solution:**

- Check file path is correct
- Make sure you installed onnxruntime: `pip install onnxruntime`
- Try: `onnxruntime-gpu` if you have NVIDIA GPU

### Issue: Wrong input shape errors

**Solution:**

- Double-check model_card.md for correct input dimensions
- Make sure you're using the right format (NCHW vs NHWC)

---

## Time Estimates

- Part 1-2 (Setup & Account): 45 min
- Part 3 (YOLOv8 Download): 45 min
- Part 4 (MoveNet Download): 45 min
- Part 5 (Testing): 1 hour
- Part 6 (Anomaly Logic): 1.5 hours
- Part 7 (Android Prep): 30 min

**Total: ~4-5 hours** (with breaks)

---

**You're all set! Once you complete this, the Android developer will have everything needed to integrate AI into the app. Good luck! 🚀**